```
From reingold@emr.cs.uiuc.edu  Tue Sep 19 12:44:28 1995
Return-Path: reingold@emr.cs.uiuc.edu
Received: from emr.cs.uiuc.edu (emr.cs.uiuc.edu [128.174.246.26]) by jcp.uchicago.edu (8
.7/8.7) with SMTP id MAA21853 for <eric@jcp.uchicago.edu>; Tue, 19 Sep 1995 12:44:27 -05
00 (CDT)
Received: from [127.0.0.1] by emr.cs.uiuc.edu with SMTP id AA19931
  (5.67b/IDA-1.5 for <eric@jcp.uchicago.edu>); Tue, 19 Sep 1995 12:44:01 -0500
Message-Id: <199509191744.AA19931@emr.cs.uiuc.edu>
To: Eric Fischer <eric@jcp.uchicago.edu>
Subject: calendar.l
Date: Tue, 19 Sep 1995 12:43:56 -0500
From: Ed Reingold <reingold@emr.cs.uiuc.edu>
Status: R

;; The following Lisp code is from ''Calendrical Calculations'' by Nachum
;; Dershowitz and Edward M. Reingold, Software---Practice & Experience,
;; vol. 20, no. 9 (September, 1990), pp. 899--928 and from
;; ''Calendrical Calculations, II: Three Historical Calendars'' by Edward M.
;; Reingold,  Nachum Dershowitz, and Stewart M. Clamen, Software---Practice
;; & Experience, vol. 23, no. 4 (April, 1993), pp. 383--404.
;;
;; This code is in the public domain, but any use of it should
;; acknowledge its source.

(defun quotient (m n)
   (floor (/ m n)))

(defun extract-month (date)
;; Month field of $date$ = (month day year).
  (first date))

(defun extract-day (date)
;; Day field of $date$ = (month day year).
  (second date))

(defun extract-year (date)
;; Year field of $date$ = (month day year).
  (third date))

(defmacro sum (expression index initial condition)
;; Sum $expression$ for $index$ = $initial$ and successive integers,
;; as long as $condition$ holds.
  (let* ((temp (gensym)))
    `(do ((,temp 0 (+ ,temp ,expression))
          (,index ,initial (1+ ,index)))
         ((not ,condition) ,temp))))

(defun last-day-of-gregorian-month (month year)
;; Last day in Gregorian $month$ during $year$.
  (if ;; February in a leap year
      (and (= month 2)
           (= (mod year 4) 0)
           (not (member (mod year 400) (list 100 200 300))))
;; Then return
      29
;; Else return
    (nth (1- month)
         (list 31 28 31 30 31 30 31 31 30 31 30 31))))

(defun absolute-from-gregorian (date)
;; Absolute date equivalent to the Gregorian $date$.
  (let* ((month (extract-month date))
         (year (extract-year date)))
;;  Return
```

```
    (+ (extract-day date)    ;; Days so far this month.
       (sum                  ;; Days in prior months this year.
        (last-day-of-gregorian-month m year) m 1 (< m month))
       (* 365 (1- year))     ;; Days in prior years.
       (quotient (1- year) 4);; Julian leap days in prior years...
       (-                    ;; ...minus prior century years...
        (quotient (1- year) 100))
       (quotient             ;; ...plus prior years divisible...
        (1- year) 400))))    ;; ...by 400.

(defun gregorian-from-absolute (date)
;; Gregorian (month day year) corresponding absolute $date$.
  (let* ((approx (quotient date 366));; Approximation from below.
         (year          ;; Search forward from the approximation.
          (+ approx
             (sum 1 y approx
                  (>= date
                      (absolute-from-gregorian
                       (list 1 1 (1+ y)))))))
         (month         ;; Search forward from January.
          (1+ (sum 1 m 1
                   (> date
                      (absolute-from-gregorian
                       (list m
                             (last-day-of-gregorian-month m year)
                             year))))))
         (day           ;; Calculate the day by subtraction.
          (- date (1- (absolute-from-gregorian
                       (list month 1 year))))))
;;  Return
    (list month day year)))

(defun Kday-on-or-before (date k)
;; Absolute date of the $k$day on or before $date$.
;; $k=0$ means Sunday, $k=1$ means Monday, and so on.
  (- date (mod (- date k) 7)))

(defun absolute-from-iso (date)
;; Absolute date equivalent to ISO $date$ = (week day year).
  (let* ((week (first date))
         (day (second date))
         (year (third date)))
;;  Return
    (+ (Kday-on-or-before
        (absolute-from-gregorian (list 1 4 year))
        1)                          ;;  Days in prior years.
       (* 7 (1- week))              ;;  Days in prior weeks this year.
       (1- day))))                  ;;  Prior days this week.

(defun iso-from-absolute (date)
;; ISO (week day year) corresponding to the absolute $date$.
  (let* ((approx
          (extract-year (gregorian-from-absolute (- date 3))))
         (year (if (>= date
                       (absolute-from-iso (list 1 1 (1+ approx))))
                ;; Then
                   (1+ approx)
                ;; Else
                   approx))
         (week (1+ (quotient
                    (- date (absolute-from-iso (list 1 1 year)))
                    7)))
         (day (if (= 0 (mod date 7))
                ;; Then
```

```
                7
             ;; Else
                (mod date 7))))
;;  Return
    (list week day year)))

(defun last-day-of-julian-month (month year)
;; Last day in Julian $month$ during $year$.
  (if ;; February in a leap year
      (and (= month 2) (= (mod year 4) 0))
;; Then return
      29
;; Else return
    (nth (1- month) (list 31 28 31 30 31 30 31 31 30 31 30 31))))

(defun absolute-from-julian (date)
;; Absolute date equivalent to Julian $date$.
  (let* ((month (extract-month date))
         (year (extract-year date)))
;;  Return
    (+ (extract-day date)    ;; Days so far this month.
       (sum                  ;; Days in prior months this year.
        (last-day-of-julian-month m year) m 1 (< m month))
       (* 365 (1- year))     ;; Days in prior years.
       (quotient (1- year) 4);; Leap days in prior years.
       -2)))                 ;; Days elapsed before absolute date 1.

(defun julian-from-absolute (date)
;; Julian (month day year) corresponding to absolute $date$.
  (let*
      ((approx      ;; Approximation from below.
        (quotient (+ date 2) 366))
       (year        ;; Search forward from the approximation.
        (+ approx
           (sum 1 y approx
                (>= date
                    (absolute-from-julian (list 1 1 (1+ y)))))))
       (month       ;; Search forward from January.
        (1+ (sum 1 m 1
                 (> date
                    (absolute-from-julian
                     (list m
                           (last-day-of-julian-month m year)
                           year))))))
       (day         ;; Calculate the day by subtraction.
        (- date (1- (absolute-from-julian (list month 1 year))))))
;;  Return
    (list month day year)))

(defun islamic-leap-year (year)
;; True if $year$ is an Islamic leap year.
  (< (mod (+ 14 (* 11 year)) 30) 11))

(defun last-day-of-islamic-month (month year)
;; Last day in $month$ during $year$ on the Islamic calendar.
  (if (or (oddp month)
          (and (= month 12) (islamic-leap-year year)))
;; Then return
      30
;; Else return
    29))

(defun absolute-from-islamic (date)
;; Absolute date equivalent to Islamic $date$.
```

```
  (let* ((month (extract-month date))
         (year (extract-year date)))
    (+ (extract-day date)      ;; Days so far this month.
       (* 29 (1- month))       ;; Days so far...
       (quotient month 2)      ;;           ...this year.
       (* (1- year) 354)       ;; Non-leap days in prior years.
       (quotient               ;; Leap days in prior years.
        (+ 3 (* 11 year)) 30)
       227014)))               ;; Days before start of calendar.

(defun islamic-from-absolute (date)
;; Islamic date (month day year) corresponding to absolute $date$.
  (if ;; Pre-Islamic date.
      (<= date 227014)
;; Then return
      (list 0 0 0)
;; Else
    (let* ((approx         ;; Approximation from below.
            (quotient (- date 227014) 355))
           (year           ;; Search forward from the approximation.
            (+ approx
               (sum 1 y approx
                    (>= date
                        (absolute-from-islamic
                         (list 1 1 (1+ y)))))))
           (month            ;; Search forward from Muharram.
            (1+ (sum 1 m 1
                     (> date
                        (absolute-from-islamic
                         (list m
                               (last-day-of-islamic-month m year)
                               year))))))
           (day              ;; Calculate the day by subtraction.
            (- date (1- (absolute-from-islamic
                         (list month 1 year))))))
      ;; Return
        (list month day year))))

(defun hebrew-leap-year (year)
;; True if $year$ is a leap year.
  (< (mod (1+ (* 7 year)) 19) 7))

(defun last-month-of-hebrew-year (year)
;; Last month of Hebrew $year$.
  (if (hebrew-leap-year year)
;; Then return
      13
;; Else return
    12))

(defun last-day-of-hebrew-month (month year)
;; Last day of $month$ in Hebrew $year$.
  (if (or (member month (list 2 4 6 10 13))
          (and (= month 12) (not (hebrew-leap-year year)))
          (and (= month 8) (not (long-heshvan year)))
          (and (= month 9) (short-kislev year)))
;; Then return
      29
;; Else return
    30))

(defun hebrew-calendar-elapsed-days (year)
;; Number of days elapsed from the Sunday prior to the start of the
;; Hebrew calendar to the mean conjunction of Tishri of Hebrew $year$.
```

```
  (let*
      ((months-elapsed
        (+
         (* 235            ;; Months in complete cycles so far.
            (quotient (1- year) 19))
         (* 12             ;; Regular months in this cycle.
            (mod (1- year) 19))
         (quotient         ;; Leap months this cycle
          (1+ (* 7 (mod (1- year) 19)))
          19)))
;;     (parts-elapsed (+ 5604 (* 13753 months-elapsed)))
;;     (day                                  ;; Conjunction day
;;      (+ 1 (* 29 months-elapsed) (quotient parts-elapsed 25920)))
;;     (parts (mod parts-elapsed 25920))     ;; Conjunction parts
;;
;;  The above lines of code are correct, but can have intermediate
;;  values that are too large for a 32-bit machine.  The following
;;  lines of code that replace them are equivalent, but avoid the
;;  problem.
;;
       (parts-elapsed
        (+ 204
           (* 793 (mod months-elapsed 1080))))
       (hours-elapsed
        (+ 5
           (* 12 months-elapsed)
           (* 793 (quotient months-elapsed 1080))
           (quotient parts-elapsed 1080)))
       (day                                  ;; Conjunction day
        (+ 1
           (* 29 months-elapsed)
           (quotient hours-elapsed 24)))
       (parts                                ;; Conjunction parts
        (+ (* 1080 (mod hours-elapsed 24))
           (mod parts-elapsed 1080)))
       (alternative-day
        (if (or
             (>= parts 19440)  ;; If new moon is at or after midday,
             (and
               (= (mod day 7) 2);; ...or is on a Tuesday...
               (>= parts 9924)  ;; at 9 hours, 204 parts or later...
               (not (hebrew-leap-year year)));; of a common year,
             (and
               (= (mod day 7) 1);; ...or is on a Monday at...
               (>= parts 16789) ;; 15 hours, 589 parts or later...
               (hebrew-leap-year;; at the end of a leap year
                (1- year))))
          ;; Then postpone Rosh HaShanah one day
            (1+ day)
          ;; Else
            day)))
    (if ;; If Rosh HaShanah would occur on Sunday, Wednesday,
        ;; or Friday
        (member (mod alternative-day 7) (list 0 3 5))
  ;; Then postpone it one (more) day and return
        (1+ alternative-day)
  ;; Else return
     alternative-day)))

(defun days-in-hebrew-year (year)
;; Number of days in Hebrew $year$.
  (- (hebrew-calendar-elapsed-days (1+ year))
     (hebrew-calendar-elapsed-days year)))
```

```
(defun long-heshvan (year)
;; True if Heshvan is long in Hebrew $year$.
  (= (mod (days-in-hebrew-year year) 10) 5))

(defun short-kislev (year)
;; True if Kislev is short in Hebrew $year$.
  (= (mod (days-in-hebrew-year year) 10) 3))

(defun absolute-from-hebrew (date)
;; Absolute date of Hebrew $date$.
  (let* ((month (extract-month date))
         (day (extract-day date))
         (year (extract-year date)))
;; Return
    (+ day                              ;; Days so far this month.
       (if ;; before Tishri
           (< month 7)
     ;; Then add days in prior months this year before and
     ;; after Nisan.
           (+ (sum (last-day-of-hebrew-month m year)
                   m 7 (<= m (last-month-of-hebrew-year year)))
              (sum (last-day-of-hebrew-month m year)
                   m 1 (< m month)))
     ;; Else add days in prior months this year
           (sum (last-day-of-hebrew-month m year) m 7 (< m month)))
    (hebrew-calendar-elapsed-days year);; Days in prior years.
    -1373429)))                 ;; Days elapsed before absolute date 1.

(defun hebrew-from-absolute (date)
;; Hebrew (month day year) corresponding to absolute $date$.
  (let* ((approx    ;; Approximation from below.
          (quotient (+ date 1373429) 366))
         (year      ;; Search forward from the approximation.
          (+ approx (sum 1 y approx
                         (>= date
                             (absolute-from-hebrew
                              (list 7 1 (1+ y)))))))
         (start     ;; Starting month for search for month.
          (if (< date (absolute-from-hebrew (list 1 1 year)))
              ;; Then start at Tishri
                  7
              ;; Else start at Nisan
                  1))
         (month     ;; Search forward from either Tishri or Nisan.
          (+ start
             (sum 1 m start
                  (> date
                     (absolute-from-hebrew
                      (list m
                            (last-day-of-hebrew-month m year)
                            year))))))
         (day       ;; Calculate the day by subtraction.
          (- date (1- (absolute-from-hebrew (list month 1 year))))))
;; Return
    (list month day year)))

(defun independence-day (year)
;; Absolute date of American Independence Day in Gregorian $year$.
  (absolute-from-gregorian (list 7 4 year)))

(defun Nth-Kday (n k month year)
;; Absolute date of the $n$th $k$day in Gregorian $month$, $year$.
;; If $n$<0, the $n$th $k$day from the end of month is returned
;; (that is, -1 is the last $k$day, -2 is the penultimate $k$day,
```

```
;; and so on).  $k=0$ means Sunday, $k=1$ means Monday, and so on.
  (if (> n 0)
;; Then return
      (+ (Kday-on-or-before             ;; First $k$day in month.
          (absolute-from-gregorian
           (list month 7 year)) k)
         (* 7 (1- n)))                  ;; Advance $n-1$ $k$days.
;; Else return
      (+ (Kday-on-or-before             ;; Last $k$day in month.
          (absolute-from-gregorian
           (list month
                 (last-day-of-gregorian-month month year)
                year))
           k)
         (* 7 (1+ n)))))                ;; Go back $-n-1$ $k$days.

(defun labor-day (year)
;; Absolute date of American Labor Day in Gregorian $year$.
  (Nth-Kday 1 1 9 year));; First Monday in September.

(defun memorial-day (year)
;; Absolute date of American Memorial Day in Gregorian $year$.
  (Nth-Kday -1 1 5 year));; Last Monday in May.

(defun daylight-savings-start (year)
;; Absolute date of the start of American daylight savings time
;; in Gregorian $year$.
  (Nth-Kday 1 0 4 year));; First Sunday in April.

(defun daylight-savings-end (year)
;; Absolute date of the end of American daylight savings time
;; in Gregorian $year$.
  (Nth-Kday -1 0 10 year));; Last Sunday in October.

(defun christmas (year)
;; Absolute date of Christmas in Gregorian $year$.
  (absolute-from-gregorian (list 12 25 year)))

(defun advent (year)
;; Absolute date of Advent in Gregorian $year$.
  (Kday-on-or-before (absolute-from-gregorian (list 12 3 year)) 0))

(defun epiphany (year)
;; Absolute date of Epiphany in Gregorian $year$.
  (+ 12 (christmas year)))

(defun eastern-orthodox-christmas (year)
;; List of zero or one absolute dates of Eastern Orthodox
;; Christmas in Gregorian $year$.
  (let* ((jan1 (absolute-from-gregorian (list 1 1 year)))
         (dec31 (absolute-from-gregorian (list 12 31 year)))
         (y (extract-year (julian-from-absolute jan1)))
         (c1 (absolute-from-julian (list 12 25 y)))
         (c2 (absolute-from-julian (list 12 25 (1+ y)))))
   (append
    (if ;; c1 occurs in current year
        (<= jan1 c1 dec31)
;;   Then that date; otherwise, none
        (list c1) nil)
    (if ;; c2 occurs in current year
        (<= jan1 c2 dec31)
;;   Then that date; otherwise, none
        (list c2) nil))))
```

```
(defun nicaean-rule-easter (year)
;; Absolute date of Easter in Julian $year$, according to the rule
;; of the Council of Nicaea.
  (let* ((shifted-epact ;; Age of moon for April 5.
          (mod (+ 14
                  (* 11 (mod year 19)))
               30))
         (paschal-moon  ;; Day after full moon on or after March 21.
          (- (absolute-from-julian (list 4 19 year))
             shifted-epact)))
;; Return the Sunday following the Paschal moon
    (Kday-on-or-before (+ paschal-moon 7) 0)))

(defun easter (year)
;; Absolute date of Easter in Gregorian $year$.
  (let* ((century (1+ (quotient year 100)))
         (shifted-epact        ;; Age of moon for April 5...
          (mod
           (+ 14 (* 11 (mod year 19));;     ...by Nicaean rule
              (-      ;; ...corrected for the Gregorian century rule
               (quotient (* 3 century) 4))
              (quotient;; ...corrected for Metonic cycle inaccuracy.
               (+ 5 (* 8 century)) 25)
              (* 30 century));;              Keeps value positive.
           30))
         (adjusted-epact       ;;  Adjust for 29.5 day month.
          (if (or (= shifted-epact 0)
                  (and (= shifted-epact 1) (< 10 (mod year 19))))
        ;; Then
              (1+ shifted-epact)
        ;; Else
            shifted-epact))
         (paschal-moon;; Day after full moon on or after March 21.
          (- (absolute-from-gregorian (list 4 19 year))
             adjusted-epact)))
;; Return the Sunday following the Paschal moon.
    (Kday-on-or-before (+ paschal-moon 7) 0)))

(defun pentecost (year)
;; Absolute date of Pentecost in Gregorian $year$.
  (+ 49 (easter year)))

(defun islamic-date (month day year)
;; List of the absolute dates of Islamic $month$, $day$
;; that occur in Gregorian $year$.
  (let* ((jan1 (absolute-from-gregorian (list 1 1 year)))
         (dec31 (absolute-from-gregorian (list 12 31 year)))
         (y (extract-year (islamic-from-absolute jan1)))
;; The possible occurrences in one year are
         (date1 (absolute-from-islamic (list month day y)))
         (date2 (absolute-from-islamic (list month day (1+ y))))
         (date3 (absolute-from-islamic (list month day (+ 2 y)))))
;; Combine in one list those that occur in current year
    (append
      (if (<= jan1 date1 dec31)
          (list date1) nil)
      (if (<= jan1 date2 dec31)
          (list date2) nil)
      (if (<= jan1 date3 dec31)
          (list date3) nil))))

(defun mulad-al-nabi (year)
;; List of absolute dates of Mulad-al-Nabi occurring in
;; Gregorian $year$.
```

```
  (islamic-date 3 12 year))

(defun yom-kippur (year)
;; Absolute date of Yom Kippur occurring in Gregorian $year$.
  (absolute-from-hebrew (list 7 10 (+ year 3761))))

(defun passover (year)
;; Absolute date of Passover occurring in Gregorian $year$.
  (absolute-from-hebrew (list 1 15 (+ year 3760))))

(defun purim (year)
;; Absolute date of Purim occurring in Gregorian $year$.
  (absolute-from-hebrew
    (list
      (last-month-of-hebrew-year (+ year 3760));; Adar or Adar II
      14
      (+ year 3760))))

(defun ta-anit-esther (year)
;; Absolute date of Ta'anit Esther occurring in Gregorian $year$.
  (let* ((purim-date (purim year)))
    (if ;; Purim is on Sunday
        (= (mod purim-date 7) 0)
  ;; Then return prior Thursday
        (- purim-date 3)
  ;; Else return previous day
      (1- purim-date))))

(defun tisha-b-av (year)
;; Absolute date of Tisha B'Av occurring in Gregorian $year$.
  (let* ((ninth-of-av
          (absolute-from-hebrew (list 5 9 (+ year 3760)))))
    (if ;; Ninth of Av is Saturday
        (= (mod ninth-of-av 7) 6)
  ;; Then return the next day
        (1+ ninth-of-av)
  ;; Else return
      ninth-of-av)))

(defun hebrew-birthday (birthdate year)
;; Absolute date of the anniversary of Hebrew $birthdate$
;; occurring in Hebrew $year$.
  (let* ((birth-day (extract-day birthdate))
         (birth-month (extract-month birthdate))
         (birth-year (extract-year birthdate)))
    (if ;; It's Adar in a normal year or Adar II in a leap year,
        (= birth-month (last-month-of-hebrew-year birth-year))
  ;; Then use the same day in last month of $year$.
        (absolute-from-hebrew
         (list (last-month-of-hebrew-year year) birth-day year))
  ;; Else use the normal anniversary of the birth date,
  ;; or the corresponding day in years without that date
      (absolute-from-hebrew (list birth-month birth-day year)))))

(defun yahrzeit (death-date year)
;; Absolute date of the anniversary of Hebrew $death$-$date$
;; occurring in Hebrew $year$.
  (let* ((death-day (extract-day death-date))
         (death-month (extract-month death-date))
         (death-year (extract-year death-date)))
    (cond
     ;; If it's Heshvan 30 it depends on the first anniversary; if
     ;; that was not Heshvan 30, use the day before Kislev 1.
     ((and (= death-month 8)
```

```
          (= death-day 30)
          (not (long-heshvan (1+ death-year))))
      (1- (absolute-from-hebrew (list 9 1 year))))
     ;; If it's Kislev 30 it depends on the first anniversary; if
     ;; that was not Kislev 30, use the day before Teveth 1.
     ((and (= death-month 9)
           (= death-day 30)
           (short-kislev (1+ death-year)))
      (1- (absolute-from-hebrew (list 10 1 year))))
     ;; If it's Adar II, use the same day in last month of
     ;; year (Adar or Adar II).
     ((= death-month 13)
      (absolute-from-hebrew
       (list (last-month-of-hebrew-year year) death-day year)))
     ;; If it's the 30th in Adar I and $year$ is not a leap year
     ;; (so Adar has only 29 days), use the last day in Shevat.
     ((and (= death-day 30)
           (= death-month 12)
           (not (hebrew-leap-year year)));; Corrected 5/19/93 by EMR
      (absolute-from-hebrew (list 11 30 year)))
     ;; In all other cases, use the normal anniversary of the
     ;; date of death.
     (t (absolute-from-hebrew
         (list death-month death-day year))))))

(defconstant mayan-days-before-absolute-zero
;; Number of days of the Mayan calendar epoch before absolute day 0,
;; according to the Goodman-Martinez-Thompson correlation.
  1137140)

(defun absolute-from-mayan-long-count (count)
;; Absolute date corresponding to the Mayan long count $count$,
;; which is a list ($baktun$ $katun$ $tun$ $uinal$ $kin$).
  (+ (* (first count)  144000);; Baktun.
     (* (second count) 7200)  ;; Katun.
     (* (third count)  360)   ;; Tun.
     (* (fourth count) 20)    ;; Uinal.
     (fifth count)            ;; Kin (days).
     (-                       ;; Days before absolute date 0.
      mayan-days-before-absolute-zero)))

(defun mayan-long-count-from-absolute (date)
;; Mayan long count date of absolute date $date$.
  (let* ((long-count (+ date mayan-days-before-absolute-zero))
         (baktun (quotient long-count 144000))
         (day-of-baktun (mod long-count 144000))
         (katun (quotient day-of-baktun 7200))
         (day-of-katun (mod day-of-baktun 7200))
         (tun (quotient day-of-katun 360))
         (day-of-tun (mod day-of-katun 360))
         (uinal (quotient day-of-tun 20))
         (kin (mod day-of-tun 20)))
    (list baktun katun tun uinal kin)))

(defun quotient (m n)
   (floor m n))

(defconstant mayan-haab-at-epoch '(8 18))

(defun mayan-haab-from-absolute (date)
;; Mayan haab date of absolute date $date$.
  (let* ((long-count (+ date mayan-days-before-absolute-zero))
         (day-of-haab
          (mod (+ long-count
```

```
                (first mayan-haab-at-epoch)
                (* 20 (1- (second mayan-haab-at-epoch))))
             365))
        (day (mod day-of-haab 20))
        (month (1+ (quotient day-of-haab 20))))
    (list day month)))

(defun mayan-haab-difference (date1 date2)
;; Number of days from Mayan haab date $date1$ to the next
;; occurrence of Mayan haab date $date2$.
  (mod (+ (* 20 (- (second date2) (second date1)))
          (- (first date2) (first date1)))
       365))

(defun mayan-haab-on-or-before (haab date)
;; Absolute date of latest date on or before absolute date $date$
;; that is Mayan haab date $haab$.
    (- date
       (mod (- date
               (mayan-haab-difference
                (mayan-haab-from-absolute 0) haab))
            365)))

(defun adjusted-mod (m n)
;; Positive remainder of $m/n$ with $n$ instead of 0.
  (1+ (mod (1- m) n)))

(defconstant mayan-tzolkin-at-epoch '(4 20))

(defun mayan-tzolkin-from-absolute (date)
;; Mayan tzolkin date of absolute date $date$.
  (let* ((long-count (+ date mayan-days-before-absolute-zero))
         (number
          (adjusted-mod (+ long-count
                           (first mayan-tzolkin-at-epoch))
                        13))
         (name
          (adjusted-mod (+ long-count
                           (second mayan-tzolkin-at-epoch))
                        20)))
    (list number name)))

(defun mayan-tzolkin-difference (date1 date2)
;; Number of days from Mayan tzolkin date $date1$ to the next
;; occurrence of Mayan tzolkin date $date2$.
  (let* ((number-difference (- (first date2) (first date1)))
         (name-difference (- (second date2) (second date1))))
    (mod (+ number-difference
            (* 13 (mod (* 3 (- number-difference name-difference))
                       20)))
         260)))

(defun mayan-tzolkin-on-or-before (tzolkin date)
;; Absolute date of latest date on or before absolute date $date$
;; that is Mayan tzolkin date $tzolkin$.
    (- date
       (mod (- date (mayan-tzolkin-difference
                     (mayan-tzolkin-from-absolute 0)
                     tzolkin))
            260)))

(defun mayan-haab-tzolkin-on-or-before (haab tzolkin date)
;; Absolute date of latest date on or before $date$ that is Mayan
;; haab date $haab$ and tzolkin date $tzolkin$; returns nil if such
```

```
;; a haab-tzolkin combination is impossible.
  (let* ((haab-difference
          (mayan-haab-difference (mayan-haab-from-absolute 0)
                                 haab))
         (tzolkin-difference
          (mayan-tzolkin-difference (mayan-tzolkin-from-absolute 0)
                                    tzolkin))
         (difference (- tzolkin-difference haab-difference)))
    (if (= (mod difference 5) 0)
        (- date
           (mod (- date
                   (+ haab-difference (* 365 difference)))
                18980))
      nil)));;  haab-tzolkin combination is impossible.

(defun french-last-day-of-month (month year)
;; Last day of {\em month, year} on the French Revolutionary calendar.
  (if (< month 13)
      30
    (if (french-leap-year year)
        6
      5)))

(defun french-leap-year (year)
;; True if {\em year} is a leap year on the French Revolutionary calendar.
  (or (member year '(3 7 11));; Actual.
      (member year '(15 20)) ;; Anticipated.
      (and (> year 20)       ;; Proposed.
           (= 0 (mod year 4))
           (not (member (mod year 400) '(100 200 300)))
           (not (= 0 (mod year 4000))))))

(defun absolute-from-french (date)
;; Absolute date of French Revolutionary {\em date}.
  (let* ((month (first date))
         (day (second date))
         (year (third date)))
    (+ 654414;; Days before start of calendar.
       (* 365 (1- year));; Days in prior years.
       ;; Leap days in prior years.
       (if (< year 20)
           (quotient year 4);; Actual and anticipated practice,
                            ;; that is, years 3, 7, 11, and 15.
         ;; Proposed rule--there were 4 leap years before year 20.
         (+ (quotient (1- year) 4)
            (- (quotient (1- year) 100))
            (quotient (1- year) 400)
            (- (quotient (1- year) 4000))))
       (* 30 (1- month));; Days in prior months this year.
       day)));; Days so far this month.

(defun french-from-absolute (date)
;; French Revolutionary date (month day year) of absolute {\em date};
;; returns nil if $date$ is before the French Revolution.
  (if (< date 654415)
      nil;; pre-French Revolutionary date.
    (let* ((approx       ;; Approximate year from below.
            (quotient (- date 654414) 366))
           (year         ;; Search forward from the approximation.
            (+ approx
               (sum 1 y approx
                 (>= date
                     (absolute-from-french (list 1 1 (1+ y)))))))
           (month        ;; Search forward from Vendemiaire.
```

```
           (1+ (sum 1 m 1
                 (> date
                    (absolute-from-french
                     (list m
                           (french-last-day-of-month m year)
                           year))))))
          (day          ;; Calculate the day by subtraction.
           (- date
              (1- (absolute-from-french (list month 1 year))))))
    (list month day year))))

(defconstant solar-sidereal-year (+ 365 279457/1080000))
(defconstant solar-month (/ solar-sidereal-year 12))
(defconstant lunar-sidereal-month (+ 27 4644439/14438334))
(defconstant lunar-synodic-month (+ 29 7087771/13358334))

(defun solar-longitude (days)
;; Mean sidereal longitude of the sun, in degrees,
;; at date and fraction of day $days$.
  (* (mod (/ days solar-sidereal-year) 1) 360))

(defun zodiac (days)
;; Zodiacal sign of the sun, as integer in range 1..12,
;; for date and fraction of day $days$.
  (1+ (quotient (solar-longitude days) 30)))

(defun old-hindu-solar-from-absolute (date)
;; Hindu solar month, day, and year of absolute date $date$.
  (let* ((hdate (+ date 1132959 1/4));;   Sunrise on Hindu date.
         (year (quotient hdate solar-sidereal-year))
         (month (zodiac hdate))
         (day (1+ (floor (mod hdate solar-month)))))
    (list month day year)))

(defun absolute-from-old-hindu-solar (date)
;; Absolute date corresponding to Hindu solar date $date$.
  (let* ((month (first date))
         (day (second date))
         (year (third date)))
    (floor (+ (* year solar-sidereal-year);; Days in elapsed years.
              (* (1- month) solar-month)  ;; In months.
              day -1/4                    ;; Whole days until midnight.
              -1132959))))                ;; Days before absolute day 0.

(defun lunar-longitude (days)
;; Mean sidereal longitude of the moon, in degrees,
;; at date and fraction of day $days$.
  (* (mod (/ days lunar-sidereal-month) 1) 360))

(defun lunar-phase (days)
;; Longitudinal distance between the sun and the moon, as an integer
;; in the range 1..30, at date and fraction of day $days$.
  (1+ (quotient
       (mod (- (lunar-longitude days) (solar-longitude days))
            360)
       12)))

(defun new-moon (days)
;; Time of the most recent mean conjunction at or before
;; date and fraction of day $days$.
  (- days (mod days lunar-synodic-month)))

(defun old-hindu-lunar-from-absolute (date)
;; Hindu lunar month, day, and year of absolute date $date$.
```

```
  (let* ((hdate (+ date 1132959))       ;; Hindu date.
         (sunrise (+ hdate 1/4))        ;; Sunrise on that day.
         (last-new-moon                 ;; Last new moon.
          (new-moon sunrise))
         (next-new-moon                 ;; Next new moon.
          (+ last-new-moon lunar-synodic-month))
         (day (lunar-phase sunrise))    ;; Day of month.
         (month                         ;; Month of lunar year.
          (adjusted-mod (1+ (zodiac last-new-moon)) 12))
         (leapmonth                     ;; If next month the same.
          (= (zodiac last-new-moon)
             (zodiac next-new-moon)))
         (next-month                    ;; Beginning of next month.
          (+ next-new-moon
                     (if leapmonth lunar-synodic-month 0)))
         (year                          ;; Solar year of next month.
          (quotient next-month solar-sidereal-year)))
    (list month leapmonth day year)))

(defun old-hindu-lunar-precedes (date1 date2)
;; True if Hindu lunar $date1$ precedes $date2$.
  (let* ((month1 (first date1))
         (month2 (first date2))
         (leap1 (second date1))
         (leap2 (second date2))
         (day1 (third date1))
         (day2 (third date2))
         (year1 (fourth date1))
         (year2 (fourth date2)))
    (or (< year1 year2)
        (and (= year1 year2)
             (or (< month1 month2)
                 (and (= month1 month2)
                      (or (and leap1 (not leap2))
                          (and (equal leap1 leap2)
                               (< day1 day2)))))))))

(defun absolute-from-old-hindu-lunar (date)
;; Absolute date corresponding to Hindu lunar date $date$;
;; returns nil if no such date exists.
  (let* ((years (fourth date))      ;; Elapsed years.
         (months (- (first date) 2));; Elapsed whole months,
               ;; minus a month's possible difference between the
               ;; solar and lunar year.
         (approx;;  Approximate date from below by adding days...
          (+ (floor (* years solar-sidereal-year)) ;;  in years,
             (floor (* months lunar-synodic-month));;  in months,
             -1132959))            ;; and before absolute date 0.
         (try
          (+ approx             ;; Search forward to correct date,
             (sum 1 i approx    ;; or just past it.
                  (old-hindu-lunar-precedes
                   (old-hindu-lunar-from-absolute i)
                   date)))))
    (if (equal (old-hindu-lunar-from-absolute try) date)
        try
      nil)));; $date$ non-existent on Hindu lunar calendar.
```